

REPORT DOCUMENTATION PAGE

AD-A259 311



Form Approved
OPM No. 0704-0188

hour per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data is burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

ORT DATE

3. REPORT TYPE AND DATES COVERED

Final:08 Oct 1992

4. TITLE AND SUBTITLE

Validation Summary Report: International Computers Limited VME Ada Compiler
VA3.10, ICL Series 39 Level 80 (Host & Target), 921008N1.11293

5. FUNDING NUMBERS

2

6. AUTHOR(S)

National Computing Centre Limited
Manchester, UNITED KINGDOM

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Computing Centre Limited
Oxford Road
Manchester M1 7ED
UNITED KINGDOM

8. PERFORMING ORGANIZATION
REPORT NUMBER

90502/82-921112

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY
REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

International Computers Limited, VME Ada Compiler VA3.10, Machester England, ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292), ACVC 1.11.

DTIC
ELECTE
DEC 17 1992
S A D

14. SUBJECT TERMS

Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val.
Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION
OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION
UNCLASSIFIED

19. SECURITY CLASSIFICATION
OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT

AVF Control Number: 90502/82-921112

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: #921008N1.11293
International Computers Limited
VME Ada Compiler VA3.10
ICL Series 39 Level 80

Prepared By:
Testing Services
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
England

Template Version 91-05-08

92-31682



92 12 16 078



Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 8 October 1992.

Compiler Name and Version: VME Ada Compiler VA3.10

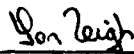
Host Computer System: ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292)


Target Computer System: ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292)

See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate #921008N1.11293 is awarded to International Computers Limited. This certificate expires 2 years after ANSI/MIL-STD-1815B is approved by ANSI.

This report has been reviewed and is approved.


Jon Leigh
Manager, System Software Testing
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
England


Ada Validation Organization
Director, Computer and Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 8 October 1992.

Compiler Name and Version: VME Ada Compiler VA3.10

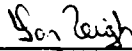
Host Computer System: ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292)

Target Computer System: ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292)


See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate #921008N1.11293 is awarded to International Computers Limited. This certificate expires 2 years after ANSI/MIL-STD-1815B is approved by ANSI.


This report has been reviewed and is approved.



Jon Leigh
Manager, System Software Testing
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
England



Ada Validation Organization
Director, Computer and Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311



Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

DECLARATION OF CONFORMANCE

The following declaration of conformance was supplied by the customer.

Declaration of Conformance

Customer: International Computers Limited

Ada Validation Facility: National Computing Centre Limited

ACVC Version: 1.11

Ada Implementation:

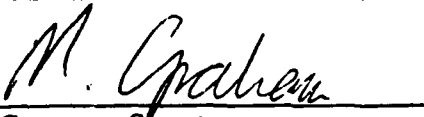
Ada Compiler Name and Version: VME Ada Compiler VA3.10

Host Computer System: ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292)

Target Computer System: ICL Series 39 Level 80 (under VME with VMEB Environment Option Version SV292)

Declaration:

I, the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A, ISO 8652-1987, FIPS 119 as tested in this validation and documented in the Validation Summary Report.


Customer Signature

8TH Oct 1992
Date

TABLE OF CONTENTS

CHAPTER 1	1
INTRODUCTION	1
1.1 USE OF THIS VALIDATION SUMMARY REPORT	1
1.2 REFERENCES	1
1.3 ACVC TEST CLASSES	2
1.4 DEFINITION OF TERMS	3
CHAPTER 2	1
IMPLEMENTATION DEPENDENCIES	1
2.1 WITHDRAWN TESTS	1
2.2 INAPPLICABLE TESTS	1
2.3 TEST MODIFICATIONS	4
CHAPTER 3	1
PROCESSING INFORMATION	1
3.1 TESTING ENVIRONMENT	1
3.2 SUMMARY OF TEST RESULTS	1
3.3 TEST EXECUTION	2
APPENDIX A	1
MACRO PARAMETERS	1
APPENDIX B	1
COMPILATION SYSTEM OPTIONS	1
APPENDIX C	1
APPENDIX F OF THE Ada STANDARD	1

CHAPTER 1

INTRODUCTION

6. The Ada implementation described above was tested according to the Ada Validation Procedures [Pro92] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro92]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311-1772

1.2 REFERENCES

- [Ada83] Reference Manual for the Ada Programming Language,
 ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
- [Pro92] Ada Compiler Validation Procedures,
 Version 3.1, Ada Joint Program Office, August 1992.
- [UG89] Ada Compiler Validation Capability User's Guide,
 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages *REPORT* and *SPPRT13*, and the procedure *CHECK_FILE* are used for this purpose. The package *REPORT* also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package *SPPRT13* is used by many tests for Chapter 13 of the Ada Standard. The procedure *CHECK_FILE* is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of *REPORT* and *CHECK_FILE* is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behaviour is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1), and possibly removing some inapplicable tests (see section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfilment of a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.

Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 AND ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro92].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 2 August 1991.

E28005C	B28006C	C32203A	C34006D	C35508I	C35508J
C35508M	C35508N	C35702A	C35702B	B41308B	C43004A
C45114A	C45346A	C45612A	C45612B	C45612C	C45651A
C46022A	B49008A	B49008B	A74006A	C74308A	B83022B
B83022H	B83025B	B83025D	C83026A	B83026B	C83041A
B85001L	C86001F	C94021A	C97116A	C98003B	BA2011A
CB7001A	CB7001B	CB7004A	CC1223A	BC1226A	CC1226B
BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E
CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C
BD3006A	BD4008A	CD4022A	CD4022D	CD4024B	CD4024C
CD4024D	CD4031A	CD4051D	CD5111A	CD7004C	ED7005D
CD7005E	AD7006A	CD7006E	AD7201A	AD7201E	CD7204B
AD7206A	BD8002A	BD8004C	CD9005A	CD9005B	CDA201E
CE2107I	CE2117A	CE2117B	CE2119B	CE2205B	CE2405A
CE3111C	CE3116A	CE3118A	CE3411B	CE3412B	CE3607B
CE3607C	CE3607D	CE3812A	CE3814A	CE3902B	

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

C24113I..N (6 tests) contain lines that exceed this implementations maximum input-line length of 126 characters.

The following 159 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113O..Y (11 tests)	C35705O..Y (11 tests)
C35706O..Y (11 tests)	C35707O..Y (11 tests)

C35708O..Y (11 tests)	C35802O..Z (12 tests)
C45241O..Y (11 tests)	C45321O..Y (11 tests)
C45421O..Y (11 tests)	C45521O..Z (12 tests)
C45524O..Z (12 tests)	C45621O..Z (12 tests)
C45641O..Y (11 tests)	C46012O..Z (12 tests)

The following 21 tests check for the predefined type `SHORT_INTEGER`; for this implementation, there is no such type:

C35404B	B36105C	C45231B	C45304B	C45411B
C45412B	C45502B	C45503B	C45504B	C45504E
C45611B	C45613B	C45614B	C45631B	C45632B
B52004E	C55B07B	B55B09D	B86001V	C86006D
CD7101E				

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than `INTEGER`, `LONG_INTEGER`, or `SHORT_INTEGER`; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than `FLOAT`, `LONG_FLOAT`, or `SHORT_FLOAT`; for this implementation, there is no such type.

C45423A..B (2 tests), C45523A, and C45622A check that the proper exception is raised if `MACHINE_OVERFLOW` is `TRUE` and the results of various floating-point operations lie outside the range of the base type; for this implementation, `MACHINE_OVERFLOW` is `FALSE`.

B86001Y uses the name of a predefined fixed-point type other than type `DURATION`; for this implementation, there is no such type.

C96005B uses values of type `DURATION`'s base type that are outside the range of type `DURATION`; for this implementation, the ranges are the same.

CA2009C and CA2009F are not applicable because the implementation requires that generic unit bodies be compiled together with their specifications.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

CD2B15B checks that `STORAGE_ERROR` is raised when the storage size specified for a collection is too small to hold a single value of the designated type; this implementation allocates more space than was specified by the length clause, as allowed by AI-00558.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package `MACHINE_CODE`.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

The tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	-----	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

The test listed in the following table checks the given file operation for the given combination of mode and access method; this implementation does not support this operations.

Test	File Operation	Mode	File Access Method
CE2111C	RESET	From IN_FILE to OUT_FILE	SEQUENTIAL_IO

The following 16 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; `USE_ERROR` is raised when this association is attempted.

CE2107B..E	CE2107G..H	CE2107L	CD2110B	CE2110D
CE2111D	CE2111H	CE3111B	CE3111D..E	CE3114B
CE3115A				

CE2203A checks that `WRITE` raises `USE_ERROR` if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that `WRITE` raises `USE_ERROR` if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 71 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B22003A	B22005I	B25002A	B26001A	B26002A	B26005A
B27005A	B28003A	B29001A	B33301B	B35101A	B37106A
B37301B	B37302A	B38003A	B38003B	B38009A	B38009B
B55A01A	B61001C	B61001F	B61001H	B61001I	B61001M
B61001R	B61001W	B67001H	B83A07A	B83A07B	B83A07C
B83E01C	B83E01D	B83E01E	B85001D	B85008D	B91001A
B91002A	B91002B	B91002C	B91002D	B91002E	B91002F
B91002G	B91002H	B91002I	B91002J	B91002K	B91002L
B95030A	B95061A	B95061F	B95061G	B95077A	B97103E
B97104G	BA1001A	BA1101B	BC1109A	BC1109C	BC1109D
BC1202A	BC1202E	BC1202F	BC1202G	BD2A25A	BE2210A
BE2413A					

C64103A and C95084A were graded passed by Evaluation Modification as directed by the AVO. Because this implementation's actual values for `LONG_FLOAT'SAFE_LARGE` and `SHORT_FLOAT'LAST` lie within one (`SHORT_FLOAT`) model interval of each other, the tests' floating-point applicability check may evaluate to `TRUE` and yet the subsequent expected exception need not be raised. The AVO ruled that the implementation's behaviour should be graded as passed because the implementation passed the integer and fixed-point checks; the following `REPORT.FAILED` messages were produced after the type conversions at line 198 in C64103A and lines 101 and 250 in C95084A failed to raise exceptions:

C64103A:	"EXCEPTION NOT RAISED AFTER CALL -P2 (B)"
C95084A:	"EXCEPTION NOT RAISED BEFORE CALL - T2 (A)"
	"EXCEPTION NOT RAISED AFTER CALL - T5 (B)"

C83030C and C86007A were graded passed by Test Modification as directed by the AVO. These tests were modified by inserting "PRAGMA ELABORATE (REPORT);" before the package declarations at lines 13 and 11, respectively. Without the pragma, the packages may be elaborated prior to package REPORT's body, and thus the packages' calls to function REPORT.IDENT_INT at lines 14 and 13, respectively, will raise PROGRAM_ERROR.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report, together with the following.

The memory Size of the Host/Target Configuration is 64 Mbytes.

For technical information about this Ada implementation, contact:

Christine Saunders
International Computers Limited
Eskdale Road
Winnersh
Wokingham
Berks
RG11 5TT

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro92].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system -- if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

a)	Total Number of Applicable Tests	3816	
b)	Total Number of Withdrawn Tests	95	
c)	Processed Inapplicable Tests	259	
d)	Non-Processed I/O Tests	0	
e)	Non-Processed Floating-Point Precision Tests	0	
f)	Total Number of Inapplicable Tests	259	
g)	Total Number of Tests for ACVC 1.11	4170	(a+b+f)

3.3 TEST EXECUTION

A set of magnetic tapes containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the set of magnetic tapes were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

LISTINGS selected for SOURCE and OBJECT. The other options as per default.

Test output, compiler and linker listings, and job logs were captured on a set of magnetic tapes and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for \$MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
\$MAX_IN_LEN	126 -- Value of V
\$BIG_ID1	(1..V-1 => 'A', V => '1')
\$BIG_ID2	(1..V-1 => 'A', V => '2')
\$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
\$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
\$BIG_INT_LIT	(1..V-3 => '0') & "298"
\$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
\$BIG_STRING1	"" & (1..V/2 => 'A') & ""
\$BIG_STRING2	"" & (1..V-1-V/2 => 'A') & '1' & ""
\$BLANKS	(1..V-20 => ' ')
\$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
\$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
\$MAX_STRING_LITERAL	"" & (1..V-2 => 'A') & ""

MACRO PARAMETERS

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
\$ACC_SIZE	32
\$ALIGNMENT	4
\$COUNT_LAST	131070
\$DEFAULT_MEM_SIZE	16#FFFF_FFFF#
\$DEFAULT_STOR_UNIT	8
\$DEFAULT_SYS_NAME	VME_2900
\$DELTA_DOC	2#1.0#E-63
\$ENTRY_ADDRESS	T.ENT'ADDRESS
\$ENTRY_ADDRESS1	T.ENT1'ADDRESS
\$ENTRY_ADDRESS2	T.ENT2'ADDRESS
\$FIELD_LAST	67
\$FILE_TERMINATOR	' '
\$FIXED_NAME	NO_SUCH_TYPE
\$FLOAT_NAME	NO_SUCH_TYPE
\$FORM_STRING	""
\$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
\$GREATER_THAN_DURATION	75000.0
\$GREATER_THAN_DURATION_BASE_LAST	2#1.0#E44
\$GREATER_THAN_FLOAT_BASE_LAST	8.0E+75
\$GREATER_THAN_FLOAT_SAFE_LARGE	16#0.FFFF_FFFF_FFFF_F0#E63

MACRO PARAMETERS

\$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	16#0.FFFF_FC#E63
\$HIGH_PRIORITY	63
\$ILLEGAL_EXTERNAL_FILE_NAME1	<NOT-A-VME-FILENAME>
\$ILLEGAL_EXTERNAL_FILE_NAME2	[ANOTHER-BAD-VMF-FILENAME]
\$INAPPROPRIATE_LINE_LENGTH	4096
\$INAPPROPRIATE_PAGE_LENGTH	-1
\$INCLUDE_PRAGMA1	PRAGMA INCLUDE ("A28006D1.TST")
\$INCLUDE_PRAGMA2	PRAGMA INCLUDE ("B28006D1.TST")
\$INTEGER_FIRST	-2147483648
\$INTEGER_LAST	2147483647
\$INTEGER_LAST_PLUS_1	2147483648
\$INTERFACE_LANGUAGE	S3
\$LESS_THAN_DURATION	-75000.0
\$LESS_THAN_DURATION_BASE_FIRST	-2#1.0#E45
\$LINE_TERMINATOR	''
\$LOW_PRIORITY	0
\$MACHINE_CODE_STATEMENT	NULL;
\$MACHINE_CODE_TYPE	NO_SUCH_TYPE
\$MANTISSA_DOC	63
\$MAX_DIGITS	18
\$MAX_INT	9223372036854775807

MACRO PARAMETERS

\$MAX_INT_PLUS_1	9223372036854775808
\$MIN_INT	-9223372036854775808
\$NAME	NO_SUCH_TYPE_AVAILABLE
\$NAME_LIST	VME_2900
\$NAME_SPECIFICATION1	:ADAVAL.X2120A(1,*,1)
\$NAME_SPECIFICATION2	:ADAVAL.X2120B(1,*,1)
\$NAME_SPECIFICATION3	:ADAVAL.X3119A(1,*,1)
\$NEG_BASED_INT	16#FFFF_FFFF_FFFF_FFFE#
\$NEW_MEM_SIZE	16#FFFF_FFFF#
\$NEW_STOR_UNIT	8
\$NEW_SYS_NAME	VME_2900
\$PAGE_TERMINATOR	ASCII.FF
\$RECORD_DEFINITION	NEW INTEGER;
\$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
\$TASK_SIZE	32
\$TASK_STORAGE_SIZE	8192
\$TICK	0.000002
\$VARIABLE_ADDRESS	VAR'ADDRESS
\$VARIABLE_ADDRESS1	VAR1'ADDRESS
\$VARIABLE_ADDRESS2	VAR2'ADDRESS
\$YOUR_PRAGMA	INTERFACE_SPELLING

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

D-1 - D-2 - D-3

D.3 ADA_COMPILE ADA

Summary

The ADA_COMPILE command is used to compile one or more Ada compilation units.

```
ADA_COMPILE (
    @literal@          INPUT,
    @literal@          SUBLIBRARY := "",
    @literal@          SAVELIST  := "",
    @superliteral@     LISTINGS  := SOURCE,
    @literal@          CONFIG    := "",
    @literal@          TEST      := "",
    @boolean@          PROGRESS  := FALSE,
    @int@              UNIT_ID   := -1,
    @boolean@          SOURCESAVE := TRUE,
    @literal@          SUPPRESS_CHECKS := "NO",
    @literal@          OPTIMISE   := "NO",
    @boolean@          DEBUG      := FALSE,
    @response@         RESPONSE  := RESULT)
```

Parameter details

Keyword	Use, options and effect	Default
INPUT	<p>Name of file (in any VME format) containing the input to the compiler. The file may hold one or more compilation units.</p> <p>Must not be *STDAD.</p>	Mandatory
SUBLIBRARY	<p>Specifies the current sublibrary and thereby also the current library which consists of the current sublibrary and its ancestor sublibraries (see Section 3.1). The name may be up to 16 characters.</p> <p>If the parameter is defaulted the sublibrary designated by the JSV ICL8ADASUBLIBRARY is used as the current sublibrary.</p>	Null
SAVELIST	<p>Name of permanent VME file to hold the compilation listing. This must not be greater than 38 characters.</p> <p>Alternatively the name of a library followed by a full stop may be specified. In this case the filename is the terminal part of the input file name and the complete name must not be greater than 38 characters.</p>	Null

If absent, a temporary file is created, the name of which is the source file terminal name prefixed by "ICL8ADALF". If the length of the source name is greater than 23 characters then, prior to the addition of the prefix, it will be truncated with leading characters removed as necessary.

LISTINGS	<p>Listings required: SOURCE, OBJECT, XREF.</p> <p>A value of "NONE" will suppress the compilation listing.</p>	SOURCE
CONFIG	<p>Name of VME file holding configuration information (see 4.4).</p> <p>If absent a standard configuration file is used.</p>	Null
TEST	<p>Literal controlling production of diagnostic listings and conditional compilation. This parameter should only be used at the request of the Software Products Support Unit.</p>	Null
PROGRESS	<p>Boolean controlling the issue of progress FALSE messages.</p> <p>A value of TRUE will cause messages to be generated on the user's terminal as the passes are entered.</p>	
UNIT_ID	<p>Integer controlling the unit number of the compiled program unit. (See 2.4.2.)</p> <p>If the compilation contains more than one program unit, UNIT_ID applies only to the first.</p> <p>Of use when incorporating non-Ada Code.</p> <p>The default value of -1 causes the compiler to select a suitable unit number.</p>	-1
SOURCESAVE	<p>A value of TRUE will cause the source text of the compilation unit to be stored in the program library. If the source text file has several compilation units the source text for each compilation unit will be stored.</p> <p>The source is not stored if there is an error in the compilation unit.</p>	TRUE

The source texts stored in the library can be extracted (or inspected) using the Program Library Utility.

A value of FALSE means that the source is not stored in the library.

SUPPRESS_
CHECKS

A value of "YES" specifies that no checking should be performed at run time.

"NO"

Any other value will cause code to be generated to do the checking described in the LRM.

OPTIMISE

A value of "YES" will cause various optimisations to become active (see 4.7).

"NO"

DEBUG

A value of TRUE will cause information to be generated for the debugger.

FALSE

A value of FALSE specifies that no such information should be generated.

RESPONSE

Specifies the name of a JSV which will contain the result of the command call.

RESULT

The value returned will be one of:-

0	Success
233001	An unhandled exception has occurred
233002	An abnormal termination condition has been detected
233004	ADA_COMPILE ABANDONED
233009	ERRORS DETECTED IN SOURCE TEXT

D.4 ADA_LINK

Summary

ADA_LINK is invoked to produce an executable program from the current program library.

```
ADA_LINK (
    @literal@    MAINUNIT,
    @literal@    SUBLIBRARY      :="",
    @literal@    COLLECTLIB,
    @literal@    TAG,            :="",
    @literal@    SAVELIST        :="",
    @boolean@    DEBUG          :=FALSE,
    @literal@    DETAILS        :="N",
    @literal@    EXTRACOMMANDS   :="",
    @integer@    TEST           :=0,
    @response@   RESPONSE       :=RESULT )
```

Parameter details

Keyword	Use, options and effect	Default
MAINUNIT	Specifies the main program which must be a library unit of the current library, but not necessarily of the current sublibrary.	Mandatory
SUBLIBRARY	Specifies the current sublibrary and thereby also the current library which consists of the current sublibrary and its ancestor sublibraries (see Section 3.1). The name may be up to 16 characters. If the parameter is defaulted the sublibrary designated by the JSV ICL8ADASUBLIBRARY is used as the current sublibrary.	Null
COLLECTLIB	Specifies the name of the OMF library in which the Ada program will be stored. The filename used will be:	Mandatory

<main program name>MOD

with an entry name of:

<main program name>.

TAG	This parameter is used to form the names of both the OMF library to be created to hold the Ada modules required for the name (ICL8ADAOMF<tag>) and the Collector's command file (ICL8ADAOPT<tag>). After successful collection these are deleted. They remain only if the collection fails or is not attempted.	Main program name
SAVELIST	<p>Specifies the name of the file to hold the information produced by the linker (see section 6.2).</p> <p>If absent and provided the DETAILS or TEST parameter is used a file "ICL8ADALINKLOG" will be used and automatically listed, and deleted.</p> <p>Output from the Collector is sent to a separate workfile which is deleted unless a failure occurs.</p>	Null
DEBUG	<p>A value of TRUE will cause information to be generated for the debugger.</p> <p>A value of FALSE specifies that no such information should be generated.</p>	FALSE
DETAILS	<p>Specifies the amount of information the linker will output on the optional log file.</p> <p>By default only error messages and a short summary are output. With DETAILS = "Y", "y" or any string starting with these characters more information from the linking process is output.</p> <p>A more precise description of the output is found in Section 6.2.</p>	"N"
EXTRACOMMANDS	Specifies a file containing minor commands which will be added to the command file supplied to the Collector.	Null
TEST	This parameter should only be used as advised by the Software Products Support Unit.	0

RESPONSE	Specifies the name of a JSV which will contain the result of the command call.	RESULT
The value returned will be one of:-		
0	Success	
233001	An unhandled exception has occurred	
233002	An abnormal termination condition has been detected	
233005	ADA_LINK ABANDONED	
233008	COLLECTION NOT ATTEMPTED	
Other	Failure in VME Collector	

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD are presented on the following three pages.

APPENDIX F Implementation Dependent Characteristics

The following sections describe the implementation dependent characteristics of the compiler.

Sections F2 onwards address the topics given in the Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A), except for topic (5), which is not relevant since implementation-generated names for implementation-dependent components are supported by this compiler.

F.1 Predefined Types

This section describes the implementation-dependent predefined types declared in the predefined package STANDARD (cf. [LRM] Annex C), and the relevant attributes of these types.

Integer Types

Two predefined integer types are implemented, INTEGER and LONG_INTEGER.

They have the following attributes:

INTEGER'FIRST	=	-2_147_483_648
INTEGER'LAST	=	2_147_483_647
INTEGER'SIZE	=	32

LONG_INTEGER'FIRST	=	-16:8000_0000_0000_0000:
LONG_INTEGER'LAST	=	16:7FFF_FFFF_FFFF_FFFF:
LONG_INTEGER'SIZE	=	64

Floating Point Types

Three predefined floating point types are supported, SHORT_FLOAT, FLOAT and LONG_FLOAT.

They have the following attributes:

SHORT_FLOAT'DIGITS	=	6
SHORT_FLOAT'FIRST	=	-16:0.FFFF_FF:E63
SHORT_FLOAT'LAST	=	16:0.FFFF_FF:E63
SHORT_FLOAT'SIZE	=	32
SHORT_FLOAT'SAFE_SMALL	=	2:1.0:E-253
SHORT_FLOAT'SAFE_LARGE	=	16:0.FFFF_F8:E63
SHORT_FLOAT'SAFE_EMAX	=	252
SHORT_FLOAT'MACHINE_RADIX	=	16
SHORT_FLOAT'MACHINE_MANTISSA	=	6
SHORT_FLOAT'MACHINE_EMAX	=	63
SHORT_FLOAT'MACHINE_EMIN	=	-64
SHORT_FLOAT'MACHINE_ROUNDS	=	FALSE
SHORT_FLOAT'MACHINE_OVERFLOWS	=	FALSE


```

FLOAT'DIGITS           = 15
FLOAT'FIRST            = -16:0.FFFF_FFFF_FFFF_FF:E63
FLOAT'LAST             = 16:0.FFFF_FFFF_FFFF_FF:E63
FLOAT'SIZE             = 64
FLOAT'SAFE_SMALL       = 2:1.0:E-253
FLOAT'SAFE_LARGE       = 16:0.FFFF_FFFF_FFFF_E0:E63
FLOAT'SAFE_EMAX        = 252
FLOAT'MACHINE_RADIX    = 16
FLOAT'MACHINE_MANTISSA = 14
FLOAT'MACHINE_EMAX     = 63
FLOAT'MACHINE_EMIN     = -64
FLOAT'MACHINE_ROUNDS   = FALSE
FLOAT'MACHINE_OVERFLOW = FALSE

LONG_FLOAT'DIGITS      = 18
LONG_FLOAT'FIRST       =
    -16:0.FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF:E63
LONG_FLOAT'LAST        =
    16:0.FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF:E63
LONG_FLOAT'SIZE        = 128
LONG_FLOAT'SAFE_SMALL  = 2:1.0:E-253
LONG_FLOAT'SAFE_LARGE  =
    16:0.FFFF_FFFF_FFFF_FFF8_0000_0000_0000:E63
LONG_FLOAT'SAFE_EMAX   = 252
LONG_FLOAT'MACHINE_RADIX = 16
LONG_FLOAT'MACHINE_MANTISSA = 28
LONG_FLOAT'MACHINE_EMAX = 63
LONG_FLOAT'MACHINE_EMIN = -64
LONG_FLOAT'MACHINE_ROUNDS = FALSE
LONG_FLOAT'MACHINE_OVERFLOW = FALSE

```

Fixed Point Types

Two kinds of anonymous fixed point types are supported; FIXED and LONG_FIXED occupying 32 and 64 bits respectively with the characteristics:

```

FIXED'FIRST           = -2_147_483_648.0
FIXED'LAST            = 2_147_483_647.0
FIXED'DELTA           = 1.0
FIXED'SIZE            = 32
FIXED'MACHINE_ROUNDS  = FALSE
FIXED'MACHINE_OVERFLOW = TRUE

LONG_FIXED'FIRST      = -16:8000_0000_0000_0000.0:
LONG_FIXED'LAST       = 16:7FFF_FFFF_FFFF_FFFF.0:
LONG_FIXED'DELTA      = 1.0
LONG_FIXED'SIZE       = 64
LONG_FIXED'MACHINE_ROUNDS = FALSE
LONG_FIXED'MACHINE_OVERFLOW = TRUE

```

The Type DURATION

The predefined fixed point type DURATION is supported and has the following attributes:

DURATION'FIRST	=	-2:1.0:E44
DURATION'LAST	=	16:0FFF_FFFF_FFFF.FFFFE:
DURATION'DELTA	=	2.0E-6
DURATION'SMALL	=	2:1.0:E-19
DURATION'SIZE	=	64
DURATION'MACHINE_ROUNDS	=	FALSE
DURATION'MACHINE_OVERFLOW	=	TRUE
DURATION'LARGE	=	DURATION'LAST
DURATION'FORE	=	15
DURATION'AFT	=	6
DURATION'MANTISSA	=	63
DURATION'SAFE_LARGE	=	DURATION'LARGE
DURATION'SAFE_SMALL	=	DURATION'SMALL

F.2 Pragmas

F.2.1 Language Defined Pragmas

This section lists all language defined pragmas and any restrictions on their use and effect as compared to the explanation given in the [LRM]. Pragmas which are inappropriate to the compiler are described as "Not applicable".

Pragma CONTROLLED

Not applicable.

Pragma ELABORATE

As in the [LRM].

Pragma INLINE

Pragma INLINE causes inline expansion except in the following cases:

- a) The whole body of the subprogram for which the inline expansion is wanted has not been seen. This ensures that recursive procedures cannot be inline expanded.
- b) The subprogram call appears in an expression on which conformance check may be applied, i.e. in a formal specification, in a discriminant part, or in a formal part of an entry declaration or accept statement. See the example below:

```

1 package INLINE_TEST is
2
3     function ONE return INTEGER;
4     pragma inline (ONE);
5
6 end INLINE_TEST;
7
8 package body INLINE_TEST is
9
10    function ONE return INTEGER is
11    begin
12        return 1;
13    end ONE;
14
15    procedure DEF_PARMS (PARM : INTEGER := ONE) is
16    *** 46W-0: Warning : Inline expansion of ONE not
17                                achieved here
18    begin
19        null
20    end DEF_PARMS;
21 end INLINE_TEST;
22

```

- c) The subprogram is an instantiation of the predefined generic subprograms UNCHECKED_CONVERSION or UNCHECKED_DEALLOCATION.
- d) The subprogram is declared in a generic unit. The body of that generic unit is compiled as a secondary unit in the same compilation as a unit containing a call to (an instance of) the subprogram. See the example below:

```

1 -- A compilation with three units:
2 generic
3 package G is
4     procedure P;
5     pragma inline(P);
6 end G;
7
8 package body G is
9     procedure P is
10    begin
11        null;
12    end P;
13 end G;
14
15 with G;
16 procedure EXAMPLE is
17     package N is new G;
18 begin
19     N.P;
20 *** 43W-0: Warning: Inline expansion of P not
21                                achieved here
22
23 end EXAMPLE;
24

```

- e) The subprogram is declared by a renaming declaration.
- f) The subprogram is passed as a generic actual parameter.

A warning is given if inline expansion is not achieved.

Pragma INTERFACE

Supported for S3 (see section 8.3).

Pragma LIST

As in the [LRM].

Pragma MEMORY SIZE

Not supported, cf. SYSTEM_NAME

Pragma OPTIMIZE

Not applicable.

Pragma PACK

In the absence of any other representation clauses on the type, the effect of pragma PACK on a composite type will be as follows. Note that biased representation is not used.

For arrays

- a) BOOLEAN and other bit-sized elements will be packed one per bit.
- b) other integer, fixed and enumeration types will be packed as tightly as possible (using their minimum sizes) with the proviso that no byte contains more than one element.
- c) floating point, access and task types occupy their predefined sizes
- d) array and record elements occupy their already calculated sizes, ie packed if requested on the component type, or obeying any other size specifications on the component type.

For records

- a) inter-component gaps will be less than a byte.
- b) BOOLEAN and other bit-sized components will be packed one per bit.

c) other integer, fixed and enumeration types will be packed as tightly as possible (using their minimum sizes) with the proviso that no byte contains more than one component.

d) floating-point, access and task types occupy their pre-defined sizes.

e) array and record elements occupy their already calculated sizes, ie packed if requested on the component type, or obeying any other size specifications on the component type.

If the user wants any tighter packing, this should be done by the use of other representation clauses.

If the record also has a rep clause, fields not mentioned in the rep clause will be allocated according to the above rules.

See also section F.5.6.

Pragma PAGE

As in the [LRM].

Pragma PRIORITY

Not applicable.

Pragma SHARED

Not applicable.

Pragma STORAGE UNIT

Has no effect.

Pragma SUPPRESS

The implementation only supports the following form of the pragma:

```
pragma SUPPRESS (identifier);
```

where identifier is as defined in [LRM] section 11.7. i.e., it is not possible to restrict the omission of a certain check to a specified name.

Pragma SYSTEM NAME

Not supported. The only meaningful SYSTEM_NAME is VME_2900 when using the VME Ada Compiler.

F.2.2 Implementation Defined Pragmas

The following implementation defined pragmas are supported:

Pragma INTERFACE_SPELLING

The permitted syntax is as follows:

```
pragma INTERFACE_SPELLING (name, "S3-name");
```

This is used in conjunction with pragma INTERFACE(S3) and indicates that no body exists for the Ada subprogram *name* and an S3 procedure *S3-name* will be expected by the linker.

The use of pragma INTERFACE_SPELLING will also ensure that the S3 name is a valid S3 external name (i.e. less than or equal to 32 alphanumeric characters, of which the first is alphabetic and all are upper case).

Pragma MAIN

The permitted syntax is as follows:

```
pragma MAIN (name);
```

Indicates to the System Linker that the library subprogram *name* is a main program. (see also section 6.3)

Pragma COMMON

The permitted syntax is as follows:

```
pragma COMMON (name, omf-name);
```

Indicates to the System Linker that the library unit *name* is to be a "visible" name in the OMF module *omf-name*, and can therefore be referenced from any program forming part of the "system". (see also section 6.3)

Pragma BASE

The permitted syntax is as follows:

```
pragma BASE (name);
```

This is used to extend a system definition, and *name* is the package containing the system definition or extended system definition to be extended. (see also section 6.3)

F.3 Attributes

The following implementation defined attributes are provided:

P'DESRIPTOR for a prefix P that denotes an object:

Yields the VME descriptor to the object. The value of this attribute is of the type **DESCRIPTOR** defined in the package **SYSTEM**.

The type/bound word of the descriptor has the following format:

for **INTEGER**, **FIXED**, **SHORT_FLOAT**, **TASK**, **ACCESS**, **ENUMERATION** (including **BOOLEAN** and **CHARACTER**): type = word; bound = 1

for **LONG_INTEGER**, **LONG_FIXED**, **FLOAT**: type = long word; bound = 1

for **LONG_FLOAT**: type = long long word; bound = 1

for records: type = string (not byte-vector); bound = size of record in bytes

for unpacked arrays of records: type = string; bound = record size in bytes * total number of elements

for unpacked arrays of unconstrained records: type = string; bound = (record size in bytes + size of red tape in bytes) * total number of elements

for other unpacked arrays: as above depending on element type; bound = total number of elements

for packed arrays of **BOOLEAN**: type = bit; bound = total number of elements

for packed arrays of **CHARACTER** (e.g. **STRING**): type = byte; bound = total number of elements.

P'FULL_ADDRESS for a prefix P that denotes an object:

This attribute is the same as the predefined attribute **'ADDRESS** except for objects (not parameters) which are unconstrained discriminated records, in which case this attribute yields the address of the first storage unit of the red tape area which precedes the actual record data.

P'FULL_SIZE for a prefix P that denotes an object:

This attribute is the same as the predefined attribute 'SIZE except for objects which are unconstrained discriminated records, in which case this attribute yields the total size of the record including the red tape area which precedes the actual record data.

for a prefix P that denotes a type:

This attribute yields the same value as the attribute when applied to an object of that type.

P'EBCDIC_IMAGE for a prefix P that denotes a discrete type or subtype:

The attribute is the same as the predefined attribute 'IMAGE except that the result type is the type STRING declared in package EBCDIC.

P'EBCDIC_VALUE for a prefix P that denotes a discrete type or subtype:

The attribute is the same as the predefined attribute 'VALUE except that the actual parameter must be a value of the type STRING declared in package EBCDIC.

F.4 Packages in Root Sublibrary

F.4.1 Package SYSTEM

Package SYSTEM is

```

type ADDRESS      is new INTEGER;
subtype DESCRIPTOR is LONG_INTEGER;
type RESPONSE     is new INTEGER;
subtype PRIORITY  is INTEGER range 0..63;
type NAME         is (VME_2900);
SYSTEM_NAME:      constant NAME      := VME_2900;
STORAGE_UNIT:     constant           := 8;
MEMORY_SIZE:      constant           := 2:1:E32 - 1;
MIN_INT:          constant := -16:8000_0000_0000_0000;;
MAX_INT:          constant := 16:7FFF_FFFF_FFFF_FFFF;;
MAX_DIGITS:       constant           := 18;
MAX_MANTISSA:     constant           := 63;

```

```

FINE_DELTA:       constant           := 2:1.0:E-63;

```

```

TICK:             constant           := 0.000_002;

```

```

type INTERFACE_LANGUAGE is (S3);

```

end SYSTEM;

The definitions of ADDRESS and DESCRIPTOR are liable to change in future releases and for forward compatibility should be treated as if they were private types.

F.4.2 Package EBCDIC

A following package for the purpose of manipulating textual data held in EBCDIC format is provided:

package EBCDIC is

type CHARACTER is

```
( NUL, SOH, STX, ETX, E04, HT, E06, DEL,
  E08, E09, E0A, VT, FF, CR, SO, SI,
  DLE, DC1, DC2, DC3, E14, NL, BS, E17,
  CAN, EM, E1A, E1B, FS, GS, RS, US,
  MS, MNL, VP, E23, E24, LF, ETB, ESC,
  E28, E29, E2A, E2B, E2C, ENQ, ACK, BEL,
  E30, E31, SYN, E33, E34, E35, E36, EOT,
  E38, E39, E3A, E3B, DC4, NAK, E3E, SUB,
  ' ', E41, E42, E43, E44, E45, E46, E47,
  E48, E49, '[', '.', '<', '(', '+', '!',
  '&', E51, E52, E53, E54, E55, E56, E57,
  E58, E59, ']', '$', '*', ')', ';', '^',
  '-', '/', E62, E63, E64, E65, E66, E67,
  E68, E69, '|', ' ', '&', ' ', '>', '?',
  E70, E71, E72, E73, E74, E75, E76, E77,
  E78, '\', ':', 'f', 'e', ' ', '=', '"',
  E80, 'a', 'b', 'c', 'd', 'e', 'f', 'g',
  'h', 'i', E8A, E8B, E8C, E8D, E8E, E8F,
  E90, 'j', 'k', 'l', 'm', 'n', 'o', 'p',
  'q', 'r', E9A, E9B, E9C, E9D, E9E, E9F,
  EA0, '~', 's', 't', 'u', 'v', 'w', 'x',
  'y', 'z', EAA, EAB, EAC, EAD, EAE, EAF,
  EB0, EB1, EB2, EB3, EB4, EB5, EB6, EB7,
  EB8, EB9, EBA, EBB, EBC, EBD, EBE, EBF,
  '{', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  'H', 'I', ECA, ECB, ECC, ECD, ECE, ECF,
  '}', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
  'Q', 'R', EDA, EDB, EDC, EDD, EDE, EDF,
  '\', EE1, 'S', 'T', 'U', 'V', 'W', 'X',
  'Y', 'Z', EEA, EEB, EEC, EED, EEE, EEF,
  '0', '1', '2', '3', '4', '5', '6', '7',
  '8', '9', EFA, EFB, EFC, EFD, EFE, EFF );
```

```
L_Bracket : constant CHARACTER := '[';
Exclam    : constant CHARACTER := '!';
Ampersand : constant CHARACTER := '&';
R_Bracket : constant CHARACTER := ']';
Dollar    : constant CHARACTER := '$';
Semicolon : constant CHARACTER := ';';
Circumflex : constant CHARACTER := '^';
Bar       : constant CHARACTER := '|';
Percent   : constant CHARACTER := '%';
Underline : constant CHARACTER := '_';
Query     : constant CHARACTER := '?';
Grave     : constant CHARACTER := '`';
Colon     : constant CHARACTER := ':';
Pound     : constant CHARACTER := 'f';
```

```

At_Sign      : constant CHARACTER := '@';
Quotation    : constant CHARACTER := '"';
Tilde        : constant CHARACTER := '~';
L_Brace      : constant CHARACTER := '{';
R_Brace      : constant CHARACTER := '}';
Back_Slash   : constant CHARACTER := '\';

FE0  : constant CHARACTER := BS;  -- Back Space
FE1  : constant CHARACTER := HT;  -- Horizontal
                                         Tabulate
FE2  : constant CHARACTER := LF;  -- Line Feed
FE3  : constant CHARACTER := VT;  -- Vertical Tabulate
FE4  : constant CHARACTER := FF;  -- Form Feed
FE5  : constant CHARACTER := CR;  -- Carriage Return

IS1  : constant CHARACTER := US;  -- Unit Separator;
IS2  : constant CHARACTER := RS;  -- Record Separator
IS3  : constant CHARACTER := GS;  -- Group Separator
IS4  : constant CHARACTER := FS;  -- File Separator

TC1  : constant CHARACTER := SOH; -- Start of Heading
TC2  : constant CHARACTER := STX; -- Start of Text
TC3  : constant CHARACTER := ETX; -- End of Text
TC4  : constant CHARACTER := EOT; -- End of
                                         Transmission
TC5  : constant CHARACTER := ENQ; -- ENQuiry
TC6  : constant CHARACTER := ACK; -- Acknowledge
TC7  : constant CHARACTER := DLE; -- Data Link Escape
TC8  : constant CHARACTER := NAK; -- Negative
                                         Acknowledge
TC10 : constant CHARACTER := ETB; -- End of
                                         Transmission Block

type STRING is array(POSITIVE range <>) of CHARACTER;

pragma PACK(STRING);

-- function "=" (Left, Right : STRING) return Boolean;
-- function "/=" (Left, Right : STRING) return Boolean;
-- function "<" (Left, Right : STRING) return Boolean;
-- function "<=" (Left, Right : STRING) return Boolean;
-- function ">" (Left, Right : STRING) return Boolean;
-- function ">=" (Left, Right : STRING) return Boolean;

-- function "&"(Left : STRING;
--               Right : STRING) return STRING;

-- function "&"(Left : CHARACTER;
--               Right : STRING) return STRING;

-- function "&"(Left : STRING;
--               Right : CHARACTER) return STRING;

-- function "&"(Left : CHARACTER;
--               Right : CHARACTER) return STRING;

end EBCDIC;

```

F.4.3 VME Interface Package

The VME_IF package is provided to permit Ada programs to read and write Job Space Variables, to read the real time clock and to determine the full hierarchic name of local names.

This package uses VME Compiler Target Machine (CTM) procedures and while it is intended that the following descriptions should be sufficient for most purposes the CTM Manual [CTM] may be consulted for further details.

F.4.3.1 The Ada Specification of the Package

```
package VME_IF is
```

```
  type VME_NUMERIC_TIME is
```

```
    record
```

```
      YEAR,
```

```
      MONTH,
```

```
      DAY,
```

```
      HOUR,
```

```
      MINUTE,
```

```
      SECOND,
```

```
      MSEC_10 : INTEGER;
```

```
    end record;
```

```
-- MSEC_10 is in 100ths of a second.
```

```
  subtype VME_RESULT is INTEGER;
```

```
-- zero - success.
```

```
-- 30461 - the JSV does not exist.
```

```
-- -9105 - value too small, value truncated.
```

```
-- -9110 - the JSV was created (so will not be visible  
--          on return from the Ada program).
```

```
-- -9112 - string too short, name truncated.
```

```
  subtype VME_TIME is LONG_INTEGER;
```

```
  procedure VME_NUMTIM_NOW (TIME : out VME_NUMERIC_TIME;  
                           RES  : out VME_RESULT);
```

```
-- TIME - the current date and time.
```

```
-- RES  - always zero.
```

```
  procedure VME_CPUTIME (TIME : out VME_TIME;  
                        RES  : out VME_RESULT);
```

```
-- TIME - the process time in microseconds
```

```
-- RES  - always zero.
```

```
  procedure VME_WAIT_TIME (TIME : in  INTEGER;  
                           RES  : out VME_RESULT);
```

```
-- TIME - the delay time in milliseconds (N.B. the delay  
-- will be at least TIME).
```

```
-- RES  - zero, indicating that the delay has occurred.
```

```
procedure VME_GIVE_NAME (LOCAL : in  STRING;
                        FULL  : out STRING;
                        LEN   : out INTEGER;
                        RES   : out VME_RESULT);

-- LOCAL - specifies the LOCAL name.
-- FULL  - the corresponding full hierarchic name.
-- LEN   - the length of the name in the string FULL.
-- RES   - zero or -9112.

procedure VME_READ_STRING (NAME  : in  STRING;
                          VALUE  : out STRING;
                          LEN    : out INTEGER;
                          RES    : out VME_RESULT);

-- NAME  - name of JSV.
-- VALUE - contents of JSV in upper case.
-- LEN   - the length of the contents in string VALUE.
-- RES   - zero, -9105 or 30461.

procedure VME_READ_BOOL (NAME  : in  STRING;
                        VALUE  : out BOOLEAN;
                        RES    : out VME_RESULT);

-- NAME  - name of JSV.
-- VALUE - value of JSV.
-- RES   - zero or 30461.

procedure VME_READ_INT (NAME  : in  STRING;
                       VALUE  : out LONG_INTEGER;
                       RES    : out VME_RESULT);

-- NAME  - name of JSV.
-- VALUE - value of JSV.
-- RES   - zero or 30461.

procedure VME_WRITE_STRING (NAME  : in  STRING;
                           VALUE  : in  STRING;
                           RES    : out VME_RESULT);

-- NAME  - name of JSV.
-- VALUE - value to be written to JSV.
-- RES   - zero or -9110.

procedure VME_WRITE_BOOL (NAME  : in  STRING;
                         VALUE  : in  BOOLEAN;
                         RES    : out VME_RESULT);

-- NAME  - name of JSV.
-- VALUE - value to be written to JSV.
-- RES   - zero or -9110.
```

```

procedure VME_WRITE_INT (NAME : in  STRING;
                        VALUE : in  LONG_INTEGER;
                        RES   : out VME_RESULT);

-- NAME - name of JSV.
-- VALUE - value to be written to JSV.
-- RES   - zero or -9110.

procedure VME_READ_STRING_KEEP_CASE
                        (NAME : in  STRING;
                        VALUE : out STRING;
                        LEN   : out INTEGER;
                        RES   : out VME_RESULT);

-- NAME - name of JSV.
-- VALUE - contents of JSV.
-- LEN   - the length of the contents in string VALUE.
-- RES   - zero, -9105 or 30461.

end VME_IF;

```

F.4.3.2 Using The Interface Procedures

All of these procedures return a VME RESPONSE value (in parameter RES) which should be tested for zero or any of the specific values indicated above; other positive values indicate that a VME error has occurred; other negative values indicate that a VME warning has occurred.

JSV Procedures

Job Space Variables can be used to pass information into an Ada program and to return results. Since an Ada program is bracketed by a BEGIN and END it is important to realise that any output JSV's need to be declared in the outer block prior to entering the Ada program.

The JSV procedures use either CTM_JS_READ or CTM_JS_WRITE [CTM] and automatically convert from EBCDIC to ASCII and vice versa.

Real Time Clock Procedures

The value returned by VME_CPU_TIME has an undefined base, but it is constant for all calls within one job, thus the difference between successive calls should be taken.

The real time clock procedures use the CTM_DATE_TIME, CTM_PROC_TIME and CTM_WAIT_TIME procedures [CTM].

Give Name Procedure

This uses the CTM_GIVE_NAME procedure [CTM].

F.5 Representation Clauses

F.5.1 SIZE specifications

In general, a size specification is taken to be the number of bits to be allocated to objects of the type, not an upper bound.

Integer, enumeration and fixed Types

The minimum size clause allowed for a discrete or fixed type is the smallest number of bits required to hold the range of values. If the range has no negative values then the size allowed is the smallest number of bits to hold the unsigned range. Biased representations are not used.

The maximum size clause allowed for an integer or fixed type is 64.

The maximum size clause allowed for an enumeration type is 32.

Floating Point Types

The only size clauses allowed are the sizes of the pre-defined types, ie 32, 64, or 128.

Array Types

For a constrained array, the given size must be at least as large as the (statically determinable) size that would normally be used for the array; the size clause will not cause packing.

For an unconstrained array the size must be sufficient for the largest values of all the index subtypes (again, these must be static).

Record Types

A size clause for a record does not cause packing.

The given size must be at least as large as the size that would normally be allocated for the record.

Access Types

The only size clause allowed is 32.

Task Types

The only size clause allowed is 32.

F.5.2 STORAGE SIZE specifications

Access Types

The minimum collection size allowed is 12, the number of storage units required to hold necessary housekeeping. The maximum collection size allowed is the maximum size of an area allowed by VME. The value is rounded up to a multiple of 4.

This sets the collection size, it includes space for housekeeping. The value should be stored in, or derived from, the collection housekeeping to support the attribute of the same name.

For an access type that has not been given a collection size, 'STORAGE_SIZE returns -1. This value is accepted as a collection size specification and indicates that a dynamic sized collection is to be used.

Sized collections are allocated on the auxiliary stack.

Task Types

A storage size of a task includes the space for the control stack and the space for the auxiliary stack, but doesn't include the that for any dependent tasks.

The algorithm for dividing the space between the two stacks has not been decided yet. The maximum and minimum storage sizes allowed will be affected by this decision. The maximum control stack size is 255 kbytes, the maximum auxiliary stack size is the maximum VME area size. The size will be rounded up to a multiple of 8.

This storage size value can be interrogated, and so should be saved somewhere or derived. There should be a default value and some means of setting it.

F.5.3 SMALL specifications

Any positive real value is allowed for SMALL.

If SIZE and SMALL specifications are inconsistent, ie SIZE too small, then one of them is rejected.

F.5.4 Enumeration representation clauses

The range of enumeration representation codes allowed is:

$-2^{31} \dots 2^{31}-1$

Non-contiguous enumeration representation codes are allowed.

If size and enumeration representation clauses are inconsistent, ie size too small, then one of them is rejected.

F.5.5 Record representation clauses

Composite types must start on a word boundary.

No component may be forced to start on a non-byte boundary if to do so means it would occupy all or part of more than 8 storage units, ie bytes.

e.g. type E is (E1, E2, E3);

```

for E'SIZE use 4;
type R is
  record
    A : E;
    B : LONG_INTEGER;
  end record;

```

```

for R use
  record
    A at 0 range 0 .. 3;
    B at 0 range 4 .. 67;
  end record;

```

This would be rejected as B would occupy parts of 9 storage units and start on a non-byte boundary.

The only values allowed for the alignment clause are 1, 2 and 4. If a component for which a component clause has been given has subcomponents with alignment, the 'POSITION part of the clause must agree with the highest subcomponent alignment.

F.5.6 Restrictions

There are some restrictions on the use of non-byte aligned components of records or arrays.

- Multi-dimensional arrays of non-byte aligned elements, e.g. bit, are not allowed.
- Non-byte aligned components are not allowed as scalar parameters of mode in out or out.
- Non-byte aligned components are not allowed to be renamed.
- Non-byte aligned record components are not allowed to have default initialisations on the record type declaration.

F.6 Address Clauses

Address clauses are only supported for objects. The value for an address can only be the result of an ADDRESS attribute.

Address clauses for subprograms, packages task units and entries are not supported.

F.7 Unchecked Conversion

Unchecked conversion is only allowed between objects of the same size, where size is as defined in section 8.1 but excluding any red-tape. A compilation error will be reported by the compiler if the objects have incompatible sizes.

For dynamic arrays and unconstrained records, the size check will be performed at run time. CONSTRAINT_ERROR will be raised if the check fails.

The TARGET may not be an unconstrained record. If the TARGET is an unconstrained array, it may only be a one-dimensional array of scalar type with an index type of size 32 bits. The index of the TARGET array will start from 'FIRST of the index range.

F.8 Input-Output

F.8.1 Introduction

This implementation supports all requirements of the Ada language, by providing an interface to the Series 39/VME file system.

This section describes the functional aspects of the VME file system interface, for the benefit of systems programmers that need to control VME specific Input-Output characteristics via Ada programs.

The section is organised as follows:

Subsection F8.2 discusses the requirements of Ada Input-Output systems as given in the language definition and provides answers to issues that are not precisely described in the language definition.

Subsection F8.3 describes the relation between (Ada) files and (VME) external files.

Subsection F8.4 describes the implementation dependent FORM parameter of OPEN and CREATE procedures.

The reader should be familiar with the following documents:

The Ada Language Reference Manual [LRM]
VME Programmer's Guide [VME]

F.8.2 Clarifications of Ada Input-Output Requirements

The Ada Input-Output concepts as presented in chapter 14 of the [LRM] do not constitute a complete functional specification of the Input-Output packages. Some aspects are not discussed at all, while others are deliberately left open to an implementation. These gaps are filled in below, with reference to sections of the [LRM].

The range of the type COUNT defined in package DIRECT_IO is 0..INTEGER'LAST and in TEXT_IO is 0..131070

F.8.2.1 Assumptions

- 14.2.1(15) For any RESET operation, the content of the file is not affected.
- 14.2.1(7) For sequential and direct input-output, files created by SEQUENTIAL_IO for a given type T, may be opened (and processed) by DIRECT_IO for the same type and vice-versa, if the VME RAM for this external file supports this mode of operation. In the case of SEQUENTIAL_IO access the function END_OF_FILE (14.2.2(8)) may fail to produce TRUE in the case where the file has been written at random, leaving "holes" in the file.
- 14.2.1(15) For any attempt to overwrite an existing record the replacement record must be the same size as the one being replaced.

F.8.2.2 Implementation Choices

- 14.1(1) An external file is any VME file, which may be regarded as a logical collection of records.
- 14.1(7) An external named file created on a filestore device will exist after program termination, and may later be accessed from an Ada program.
- 14.1(13) See Section F.8.3.4 File-Access and Sharing.
- 14.2.1(3) The name parameter, when non-null, must be a valid VME filename; a file with that name will then be created. For a null name parameter, a temporary, unnamed file will be created.

The form and effect of the form parameter is discussed in Section F.8.4.
- 14.2.1(13) Deletion of a file is only supported for files on a disk device, and requires delete permission to the file.
- 14.6 Package LOW_LEVEL_IO is not provided.

F.8.3 Basic File-Mapping

Basic file-mapping concerns the relationship between Ada files and (formats of) external VME files, and the strategy for accessing the external files.

Below, the default and acceptable file formats are summarised. The symbol ES is used to denote the element size, that is, the number of bytes occupied by the

element type, or, in case of a varying size type, the maximum size (which must be determinable at the point of instantiation from the value of the SIZE attribute for the element type).

For DIRECT_IO and SEQUENTIAL_IO, when a successful connection has been made to an external file, an additional check is made that the record size of the connected file is suitable for the element size. USE_ERROR is raised if the record size is unsuitable.

F.8.3.1 DIRECT IO

An element is mapped into a single record of the external file.

For CREATE the standard file description *STDDIRECT is used by default. This is acceptable provided ES = 80. If ES is not 80 then a suitable file description must be created (any description that is supported by the direct serial RAM is acceptable) and the FORM paramter must then be used to specify this file description.

For OPEN the file specified must have a description that is supported by the direct serial RAM, and a record size matching the element size.

F.8.3.2 SEQUENTIAL IO

An element is mapped into a single record of the external file.

For CREATE the standard file description *STDM is used by default. This is acceptable provided ES is less than or equal to 2036. If ES is greater than 2036 then a suitable file description must be created (any description that is supported by the serial RAM is acceptable) and the FORM paramter must then be used to specify this file description.

For OPEN the file specified must have a description that is supported by the serial RAM.

F.8.3.3 TEXT IO

Lines of text are mapped onto records of external files.

The default files provided for STANDARD_INPUT and STANDARD_OUTPUT are *STDAD and *STDOUT respectively.

For output, the following rules apply.

The Ada line terminators and file terminators are never explicitly stored. Page terminators, except the last, are mapped onto a FF character trailing the last line of the page. (In particular, an empty page (except the last) is mapped onto a single record containing only a FF character). The last page terminator in a file is never

represented in the external file. It is not possible to write records containing more than 2048 characters. That is, the maximum line length is 2047 or 2048, depending on whether a page terminator (FF character) must be written or not.

On input, a FF trailing a record indicates that the record contains the last line of a page and that at least one more page exists. The physical end of file indicates the end of the last page.

For CREATE the standard file description *STDM is used by default. This is acceptable provided ES is less than or equal to 2036. If ES is greater than 2036 and less than or equal to 2048 then a suitable file description must be created (any description that is supported by the serial RAM is acceptable) and the FORM parameter must then be used to specify this file description. Any attempt to input or output a record containing more than 2048 characters will raise a USE_ERROR exception.

For OPEN the file specified must have a description that is supported by the serial RAM.

F.8.3.4 File-Access and Sharing

In this section a characterisation of the file-access used is given.

OPEN and CREATE procedures use the normal VME defaulting mechanism to determine the exact file to open or create. The file generation number (when not specified), defaults (for OPEN) to highest existing, or (for CREATE), one higher than the highest existing or 1 when no versions exist. If an empty string is specified as name, CREATE will create a workfile.

External files will be accessed via standard VME access methods. For SEQUENTIAL_IO and TEXT_IO, any file description supported by the serial RAM is acceptable, while for DIRECT_IO, any file description supported by the direct serial RAM is acceptable.

A file opened with mode IN_FILE will allow other processes and, indeed, the current process to open and read the file (e.g. as IN_FILE in an Ada program). For INOUT_FILE or OUT_FILE, no file sharing is allowed. In particular, attempting to gain write access to such an external file by OPEN or RESET will raise USE_ERROR.

There is an absolute VME limit of 255 on the number of concurrent file connections. Since the VME System uses a number of system files the limit for an Ada program is somewhat less than this. This absolute limit is also for sequential files; the limit for index-sequential files is considerably less.

F.8.4 Form parameter

The FORM string parameter that can be supplied to any OPEN or CREATE procedure, has the intention of enabling control over external file properties such as physical organisation, allocation etc. In the present implementation, this is achieved by a combination of the name and form parameters.

Any of the following values of the FORM parameter are permitted:

- (1) Null
- (2) LOCAL
- (3) DESC=<fd>
- (4) ALL=<fa>
- (5) DESC=<fd>,ALL=<fa>
- (6) DESC=<fd>,LOCAL
- (7) ALL=<fa>,LOCAL
- (8) DESC=<fd>,ALL=<fa>,LOCAL
- (9) DESC=*STDFORM

LOCAL has the effect of causing the run time system to treat the value of the NAME parameter as a VME local name.

DESC which only has effect on a CREATE call, specifies that the pre-existing file description <fd> should be used. A new file description can be set up using the DESCRIBE_FILE command available in VME/B.

ALL which only has effect on a CREATE call, is used when the file allocation to be used is required to be different from the default allocation; in this case <fa> must specify a pre-existing file allocation.

The special form parameter "DESC=*STDFORM" when supplied to SEQUENTIAL_IO will provide a listing workfile.

All letters in the FORM parameter must be given in upper case, and only the first 32 bytes of the form parameter are analysed. If the syntax is incorrect a message will be sent to the journal but no exception will be raised. No semantic validation is carried out; the value given will be passed unmodified to VME as parameters to a file creation interface, were it may be ignored if it conflicts with information already known about the file. e.g a library cannot contain two files with different descriptions.

F.8.5 Additional I/O Packages

No additional packages are provided.